Fig. 1a

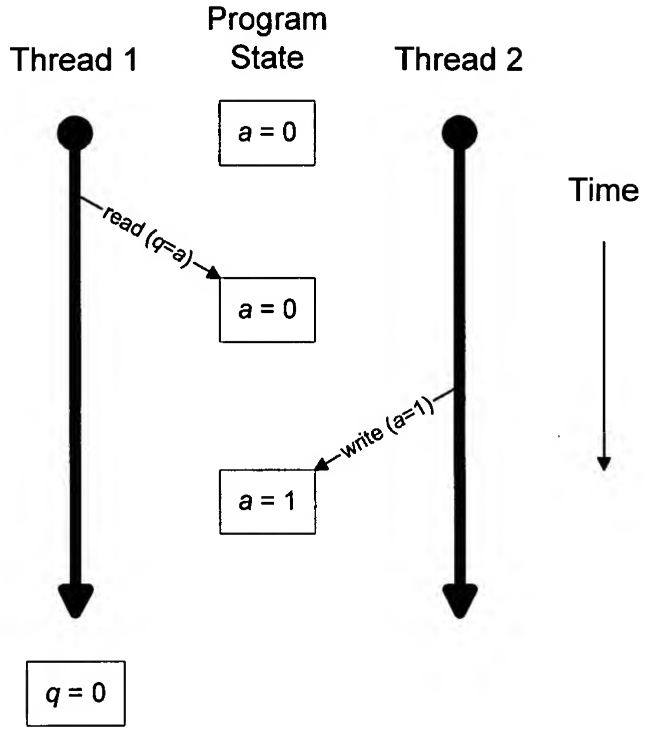

Program State
Thread 1
Thread 2
a = 0
Time
read (q=a)
a = 0
write (a=1)
a = 1
q = 0

Fig. 1b

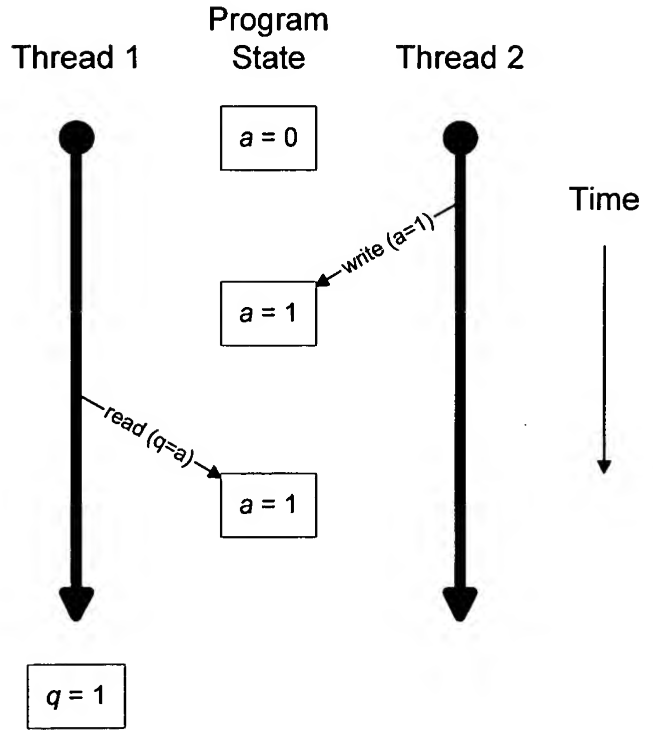

Program State
Thread 1
Thread 2
a = 0
Time
write (a=1)
a = 1
read (q=a)
a = 1
q = 1

Fig. 1c

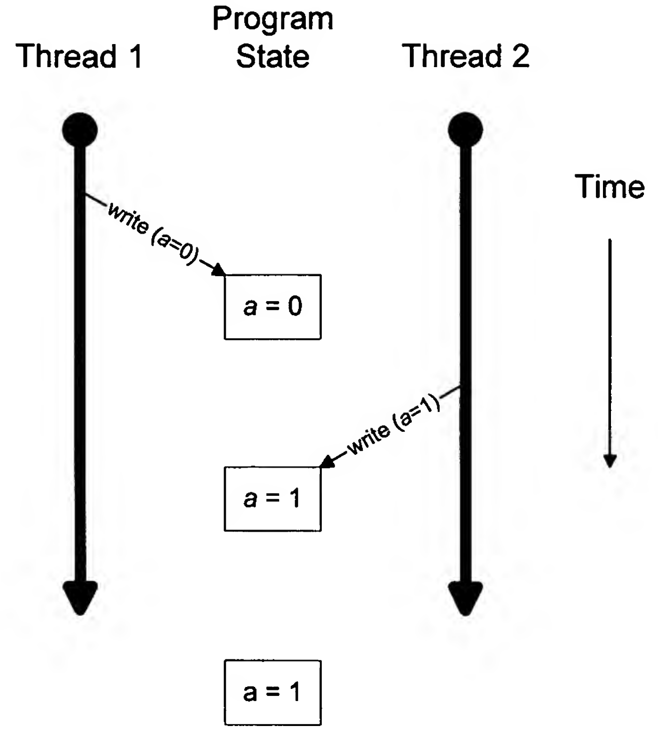

Thread 1
Program State
Thread 2
Time
write (a=0)
a = 0
write (a=1)
a = 1
a = 1

Fig. 1d

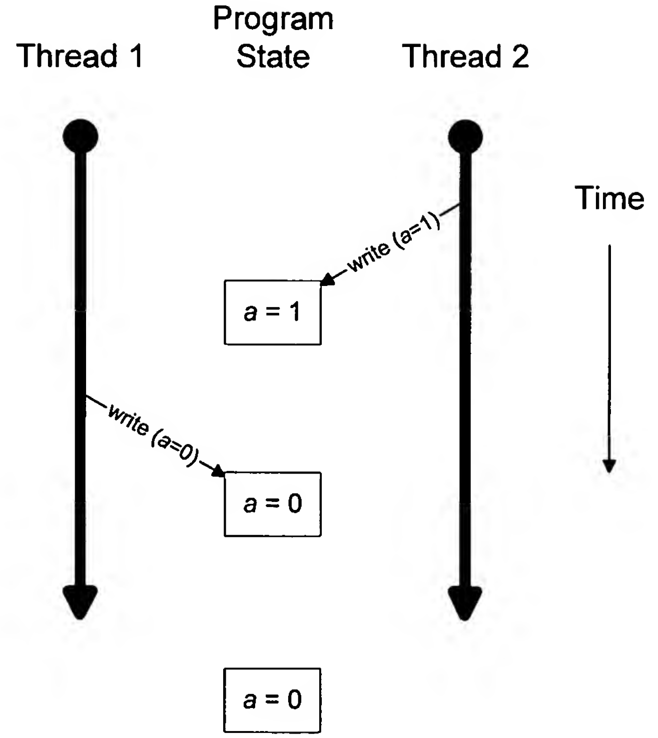

Thread 1
Program State
Thread 2
Time
write (a=1)
a = 1
write (a=0)
a = 0
a = 0

Fig. 2

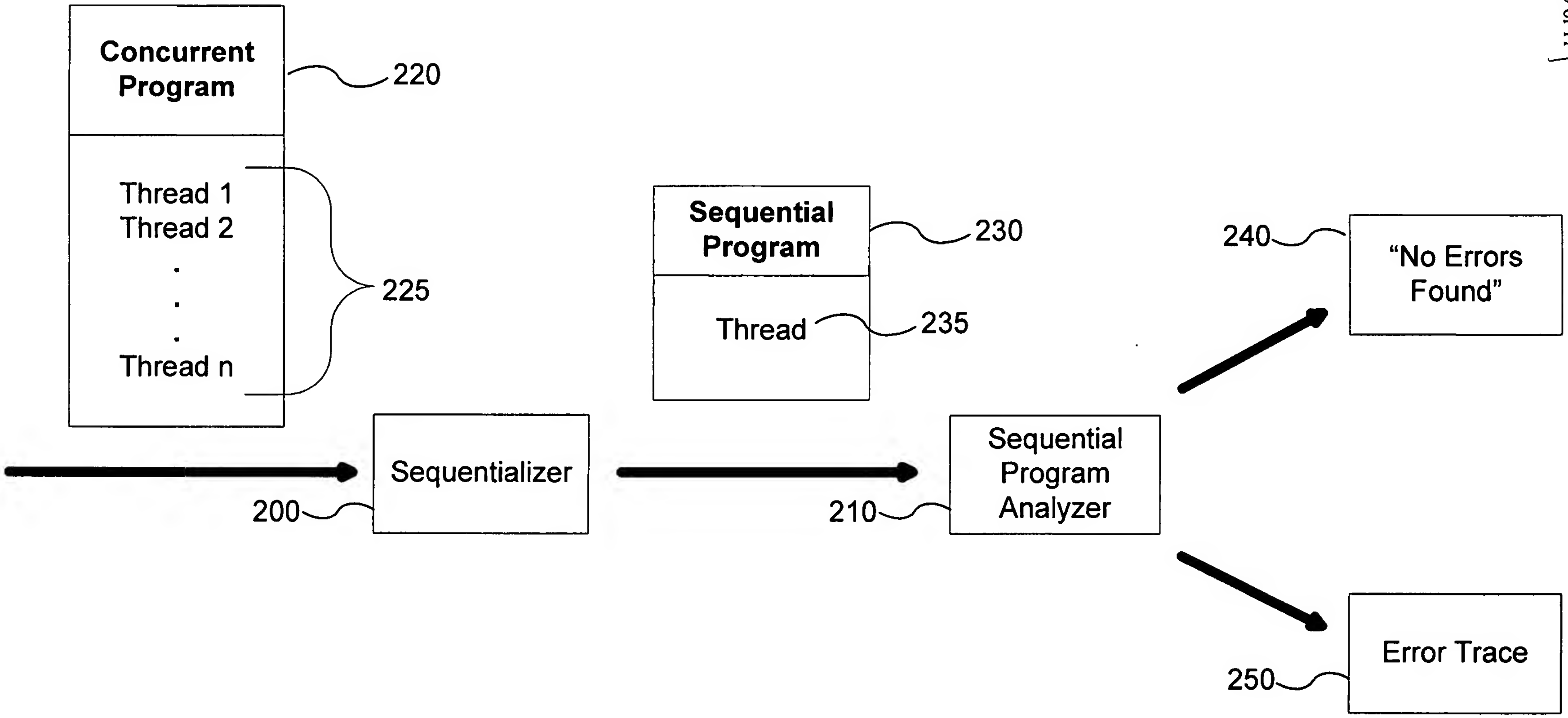
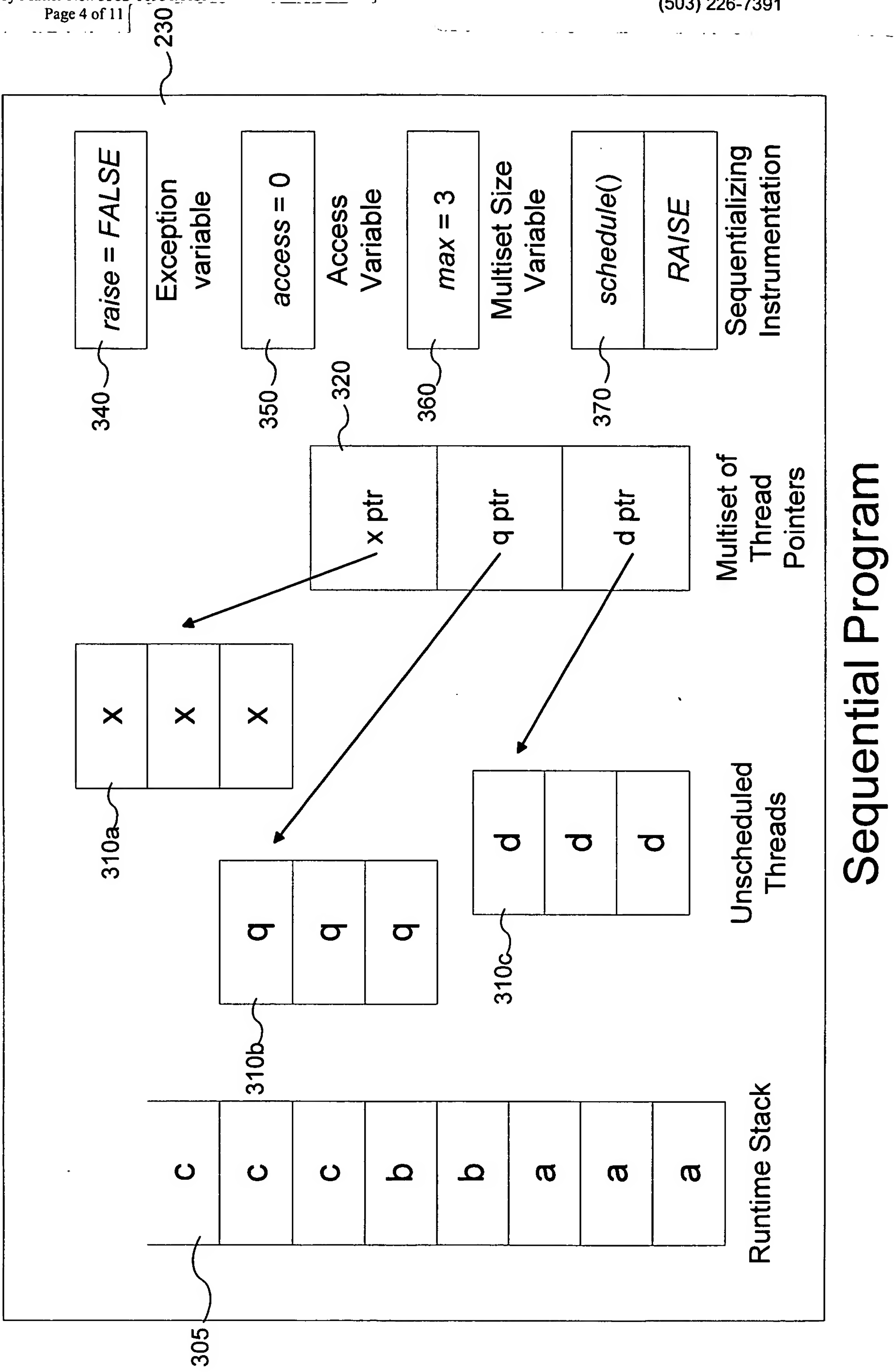

Concurrent Program
220
Thread 1
Thread 2
. . .
Thread n
225
Sequentializer
200
Sequential Program
230
Thread
235
Sequential Program Analyzer
210
"No Errors Found"
240
Error Trace
250

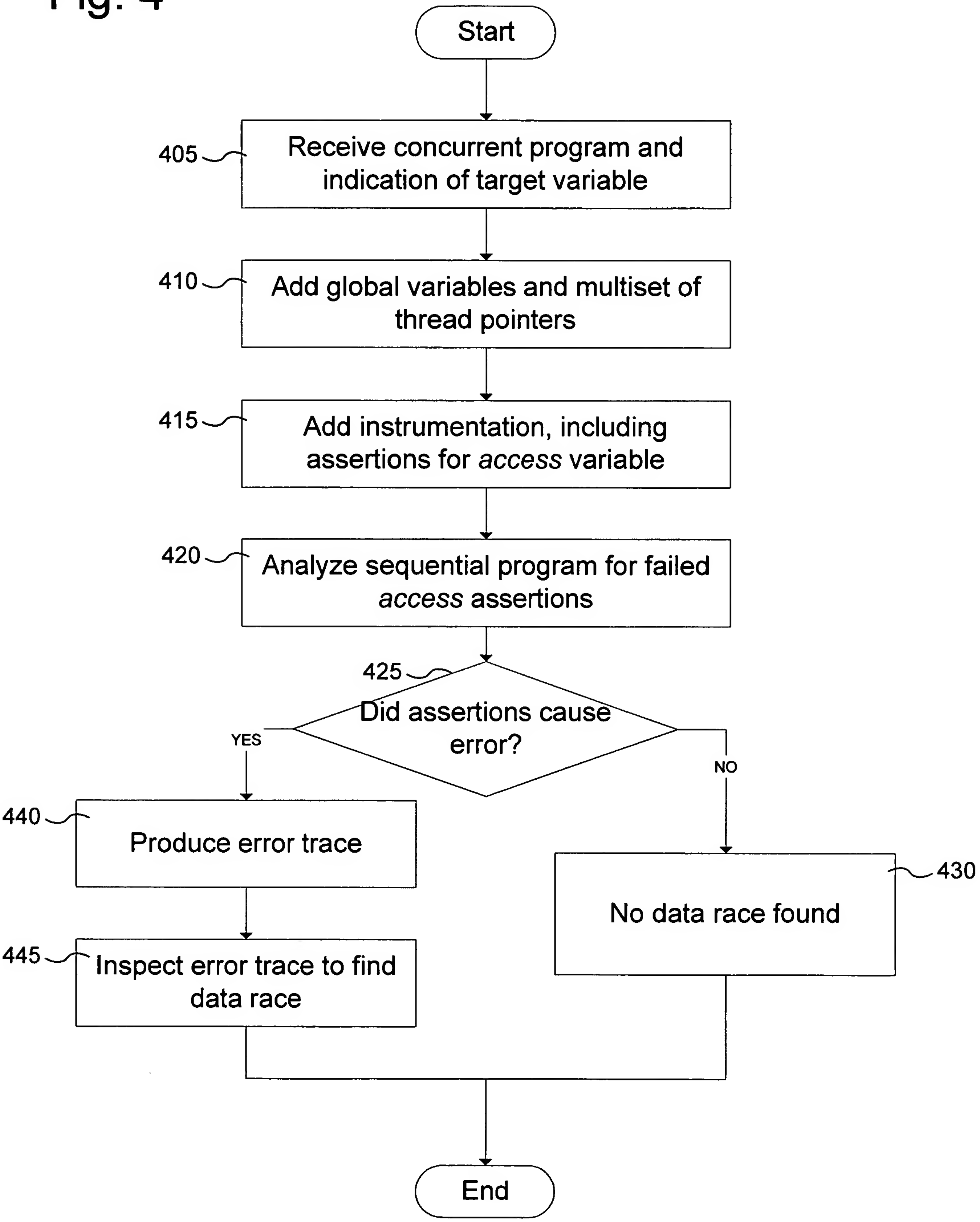
Fig. 3


230
Sequential Program
raise = FALSE
Exception variable
340
access = 0
Access Variable
350
max = 3
Multiset Size Variable
360
schedule()
RAISE
Sequentializing Instrumentation
370
320
x ptr
q ptr
d ptr
Multiset of Thread Pointers
x
x
x
310a
q
q
q
310b
d
d
d
310c
Unscheduled Threads
c
c
c
b
b
a
a
a
Runtime Stack
305

Inventor (s): Qadeer et al.
Date of Deposit: January 26, 2004
Express Mail Label No. EV 352377855 US
Title: DATA RACE DETECTION USING SEQUENTIAL PROGRAM ANALYSIS
Attorney Matter No.: 3382-66931/RCF
Page 5 of 11

Stephen A. Wight
Klarquist Sparkman, LLP
121 S.W. Salmon Street, Suite 1600
Portland, Oregon 97204
(503) 226-7391

Fig. 4



Start
405
Receive concurrent program and indication of target variable
410
Add global variables and multiset of thread pointers
415
Add instrumentation, including assertions for *access* variable
420
Analyze sequential program for failed *access* assertions
425
Did assertions cause error?
YES
NO
440
Produce error trace
445
Inspect error trace to find data race
430
No data race found
End

Fig. 5

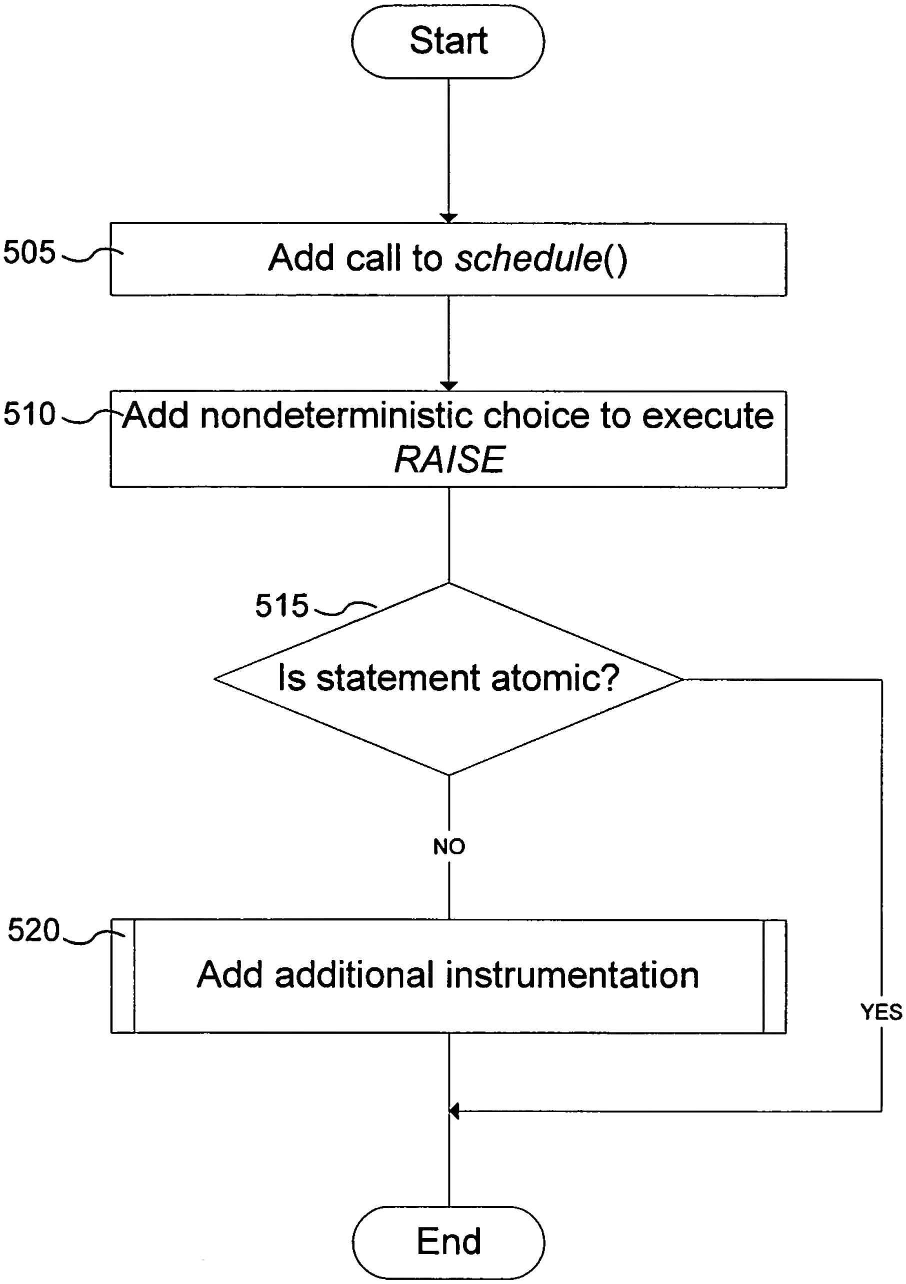

Start
505
Add call to *schedule*()
510
Add nondeterministic choice to execute *RAISE*
515
Is statement atomic?
NO
YES
520
Add additional instrumentation
End

Fig. 6

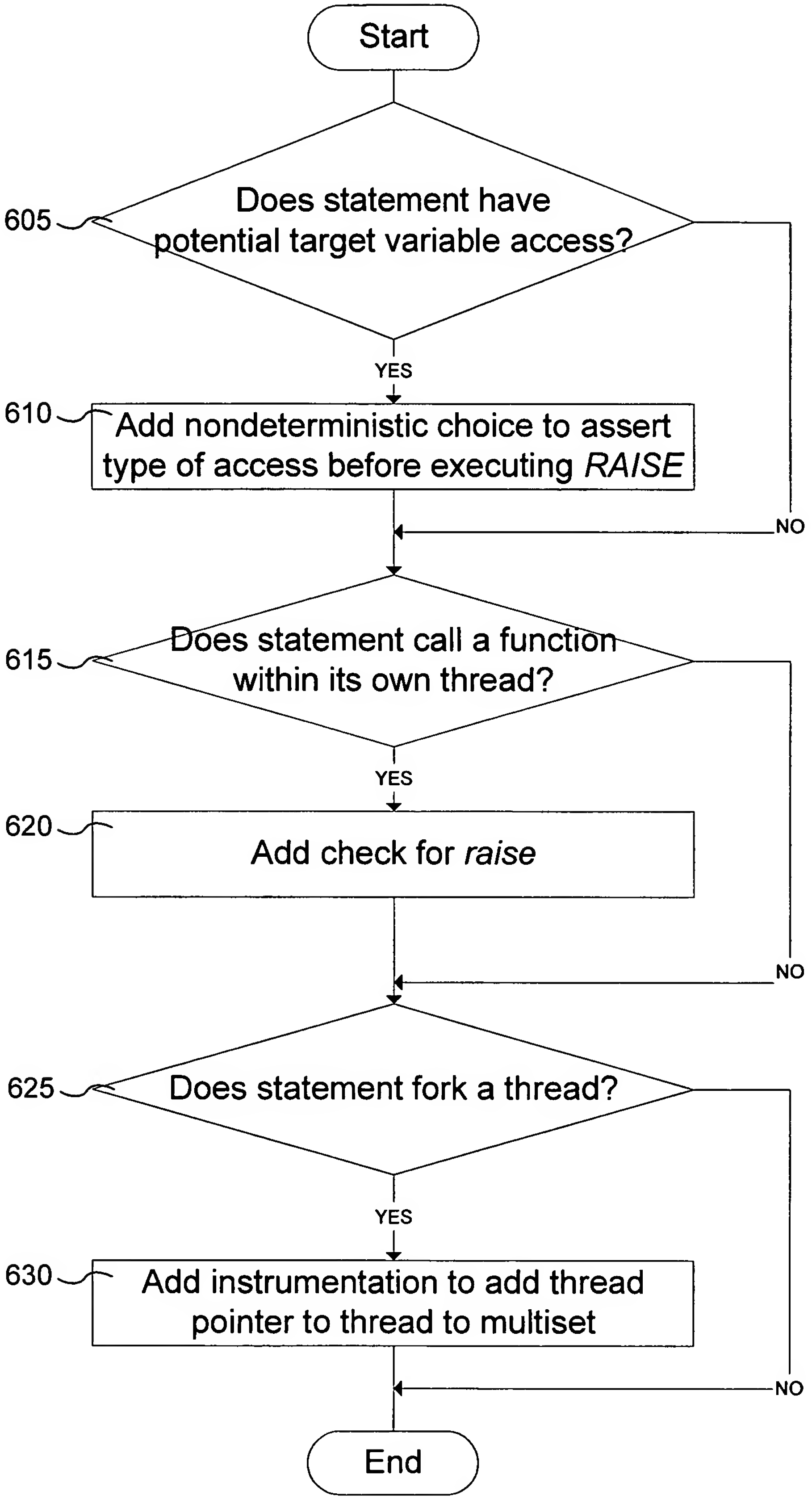

Start
605
Does statement have potential target variable access?
YES
NO
610
Add nondeterministic choice to assert type of access before executing *RAISE*
615
Does statement call a function within its own thread?
YES
NO
620
Add check for *raise*
625
Does statement fork a thread?
YES
NO
630
Add instrumentation to add thread pointer to thread to multiset
End

Fig. 7

	Original Statement		Instrumented Sequential Code
705	$v = c$	=	schedule(); choice{skip [] check$_w$(&v); RAISE}; $v = c$
710	$v = \&v_1$	=	schedule(); choice{skip [] check$_w$(&v); RAISE}; $v = \&v_1$

715	$v = {*}v_1$	=	schedule();
			choice{skip [] check$_r$(&v_1); RAISE [] check$_r$(v_1); RAISE [] check$_w$(&v); RAISE};
			$v = {*}v_1$

720	${*}v = v_1$	=	schedule(); choice{skip [] check$_r$(&v_1); RAISE [] check$_r$(&v); RAISE [] check$_w$(v); RAISE}; ${*}v = v_1$

725	$v = v_1$ op v_2	=	schedule();
			choice{skip [] check$_r$(&v_1); RAISE [] check$_r$(&v_2); RAISE [] check$_w$(&v); RAISE};
			$v = v_1$ op v_2

730	atomic {s}	=	schedule(); choice{skip [] RAISE}; s

735	$v = v_0()$	=	schedule();
			choice{skip [] check$_r$(&v_0); RAISE [] check$_w$(&v); RAISE};
			$v = v_0()$;
			if (raise) return

740	async $v_0()$	=	schedule();
			choice{skip [] check$_r$(&v_0); RAISE};
			if (size() < max) put(v_0);
			else {v_0(); raise = FALSE}

745	return	=	schedule(); return

Code Instrumentation Examples

Fig. 8a

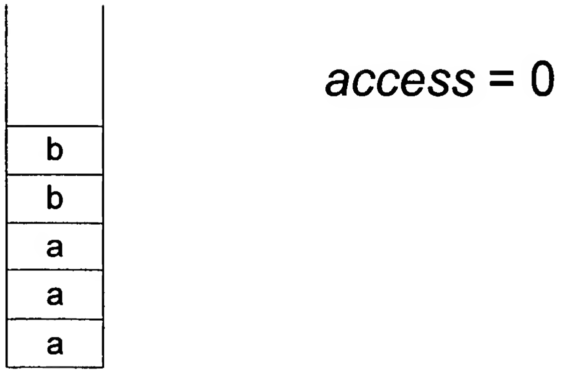


Fig. 8b

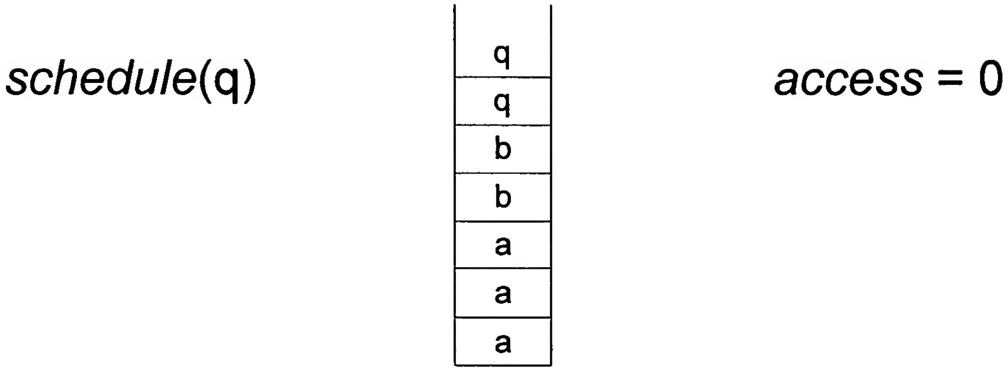


Fig. 8c

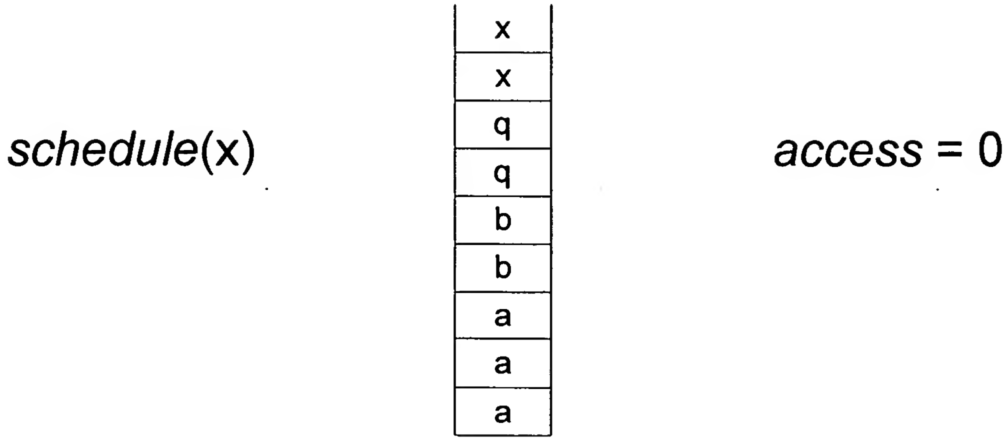


Fig. 8d

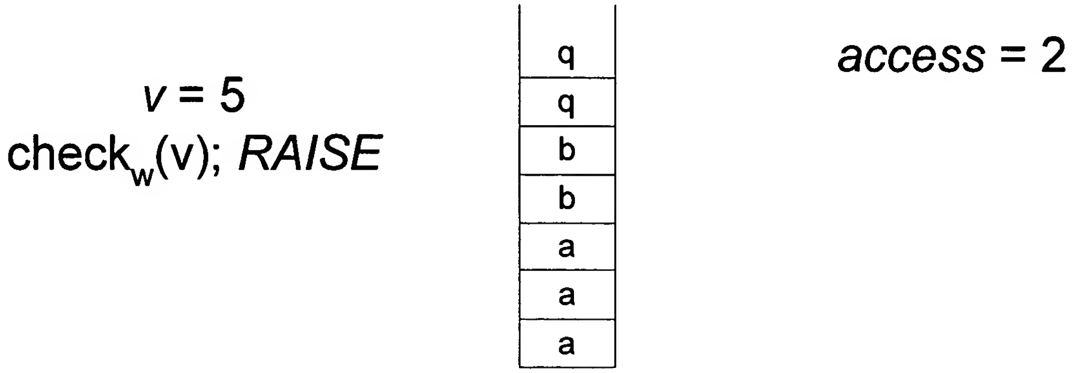


Fig. 8e

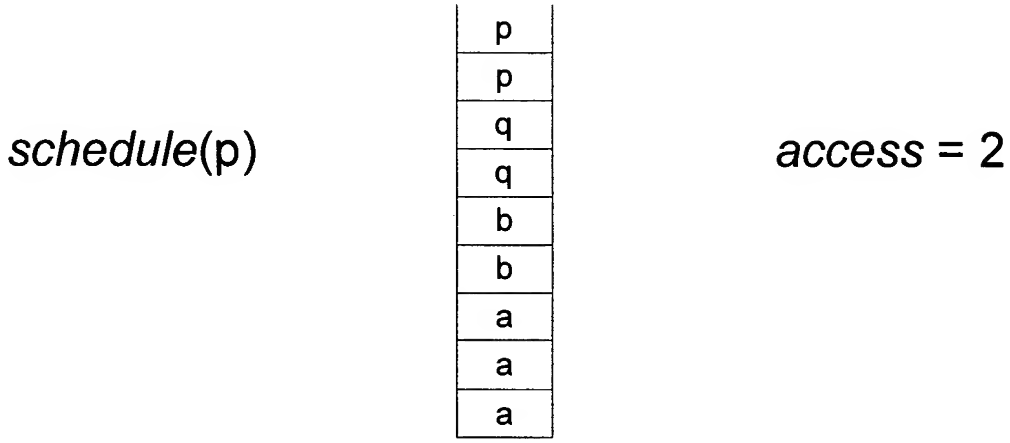


Fig. 8f

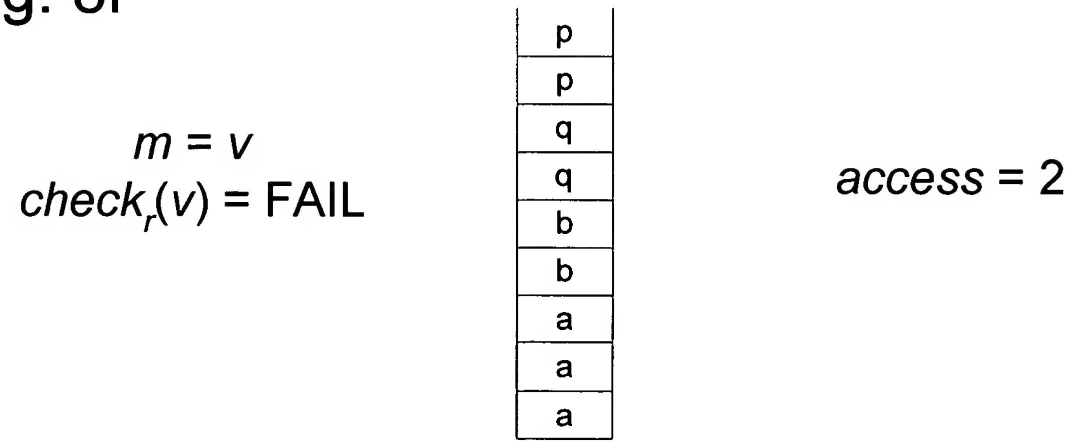
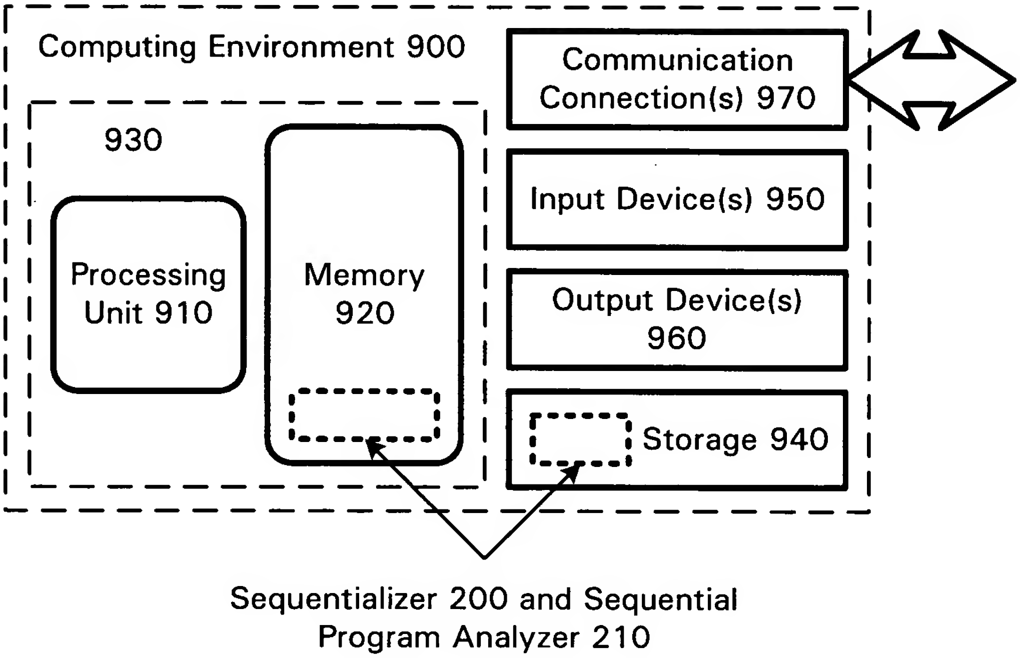


Fig. 9


Computing Environment 900
930
Processing Unit 910
Memory 920
Communication Connection(s) 970
Input Device(s) 950
Output Device(s) 960
Storage 940
Sequentializer 200 and Sequential Program Analyzer 210